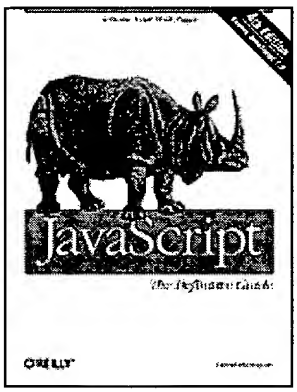
This is **G o o g l e**'s cache of http://www.oreilly.com/catalog/jscript4/chapter/ch17.html.
G o o g l e's cache is the snapshot that we took of the page as we crawled the web.
The page may have changed since that time. Click here for the current page without highlighting.
To link to or bookmark this page, use the following url: `http://www.google.com/search?q=cache:2GiVfxHR5mkC:www.oreilly.com/catalog/jscript4/chapter/ch17.html+%22dom+tree%22+%22replace+the+node%22&hl=en&ie=UTF-8`

Google is not affiliated with the authors of this page nor responsible for its content.

These search terms have been **dom tree replace the node**

O'REILLY Online Catalog

PRODUCT INDEX
SEARCH THE CATALOG



JavaScript: The Definitive Guide, 4th Edition

By David Flanagan
4th Edition November 2001
0-596-00048-0, Order Number: 0480
936 pages, $44.95

Chapter 17: The Document Object Model

A *document object model* (DOM) is an application programming interface (API) for representing a document (such as an HTML document) and accessing and manipulating the various elements (such as HTML tags and strings of text) that make up that document. JavaScript-enabled web browsers have always defined a document object model; a web-browser DOM may specify, for example, that the forms in an HTML document are accessible through the `forms[]` array of the Document object.

In this chapter, we'll discuss the W3C DOM, a standard document object model defined by the World Wide Web Consortium and implemented (at least partially) by Netscape 6 and Internet Explorer 5 and 6. This DOM standard[1] is a full-featured superset of the traditional web-browser DOM. It represents HTML (and XML) documents in a tree structure and defines properties and methods for traversing the tree and examining and modifying its nodes. Other portions of the standard specify techniques for defining event handlers for the nodes of a document, working with the style sheets of a document, and manipulating contiguous ranges of a document.

This chapter begins with an overview of the DOM standard and then describes the core portions of the standard for working with HTML documents. The discussion of the core standard is followed by short sections that explain the DOM-like features of Internet

Explorer 4 and Netscape 4. The chapter ends with an overview of two optional parts of the DOM standard that are closely related to the core. Later chapters cover advanced DOM features for working with style sheets and events.

An Overview of the DOM

The DOM API is not particularly complicated, but before we can begin our discussion of programming with the DOM, there are a number of things you should understand about the DOM architecture.

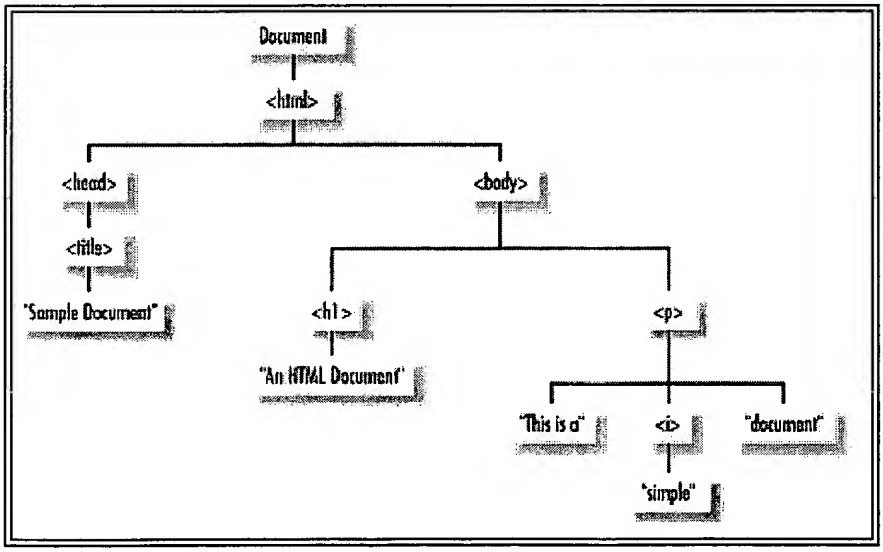
Representing Documents as Trees

HTML documents have a hierarchical structure that is represented in the DOM as a tree structure. The nodes of the tree represent the various types of content in a document. The tree representation of an HTML document primarily contains nodes representing elements or tags such as `<body>` and `<p>` and nodes representing strings of text. An HTML document may also contain nodes representing HTML comments.[2] Consider the following simple HTML document:

```
<html>
  <head>
    <title>Sample Document</title>
  </head>
  <body>
    <h1>An HTML Document</h2>
    <p>This is a <i>simple</i> document.
  </body>
</html>
```

The DOM representation of this document is the tree pictured in Figure 17-1.

Figure 17-1. The tree representation of an HTML document



If you are not already familiar with tree structures in computer programming, it is helpful to know that they borrow terminology from family trees. The node directly above a node is the

parent of that node. The nodes one level directly below another node are the *children* of that node. Nodes at the same level, and with the same parent, are *siblings*. The set of nodes any number of levels below another node are the *descendants* of that node. And the parent, grandparent, and all other nodes above a node are the *ancestors* of that node.

Nodes

The **DOM tree** structure illustrated in Figure 17-1 is represented as a tree of various types of Node objects. The Node interface[3] defines properties and methods for traversing and manipulating the tree. The `childNodes` property of a Node object returns a list of children of the node, and the `firstChild`, `lastChild`, `nextSibling`, `previousSibling`, and `parentNode` properties provide a way to traverse the tree of nodes. Methods such as `appendChild( )`, `removeChild( )`, `replaceChild( )`, and `insertBefore( )` enable you to add and remove nodes from the document tree. We'll see examples of the use of these properties and methods later in this chapter.

Types of nodes

Different types of nodes in the document tree are represented by specific subinterfaces of Node. Every Node object has a `nodeType` property that specifies what kind of node it is. If the `nodeType` property of a node equals the constant `Node.ELEMENT_NODE`, for example, know the Node object is also an Element object and you can use all the methods and properties defined by the Element interface with it. Table 17-1 lists the node types commonly encountered in HTML documents and the `nodeType` value for each one.

Table 17-1: Common node types

Interface	nodeType constant	nodeType value
Element	Node.ELEMENT_NODE	1
Text	Node.TEXT_NODE	3
Document	Node.DOCUMENT_NODE	9
Comment	Node.COMMENT_NODE	8
DocumentFragment	Node.DOCUMENT_FRAGMENT_NODE	11
Attr	Node.ATTRIBUTE_NODE	2

The Node at the root of the **DOM tree** is a Document object. The `documentElement` property of this object refers to an Element object that represents the root element of the document. For HTML documents, this is the `<html>` tag that is either explicit or implicit in the document. (The Document node may have other children, such as Comment nodes, in addition to the root element.) The bulk of a **DOM tree** consists of Element objects, which represent tags such as `<html>` and `<i>`, and Text objects, which represent strings of text. If the document parser preserves comments, those comments are represented in the **DOM tree** by Comment objects. Figure 17-2 shows a partial class hierarchy for these and other core DOM interfaces.

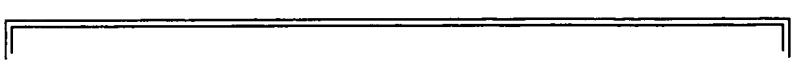



Figure 17-2. A partial class hierarchy of the core DOM API